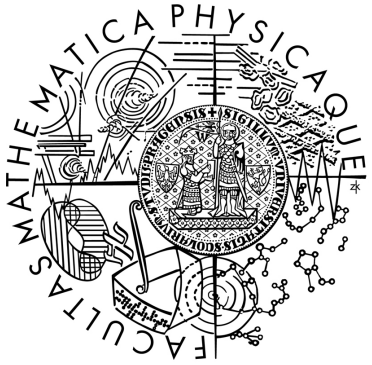
Charles University in Prague
Faculty of Mathematics and Physics

BACHELOR THESIS



Anna Roubíčková

Combining bracketed pictures

Department of Software Engineering

Supervisor: Mgr. Viliam Holub
Specialization: Computer Science

2007

I would like to thank to my supervisor Mgr. Viliam Holub for his patience and valuable advice, to Jan Pazdera, who provided me with outputs of Photoshop, to Eva Poštulková for her help with drawing sketches presented in the thesis and last but not least to Tomáš Ostatnický, who enlightened me the physical background of the problem.

I hereby certify that I wrote the thesis myself, using only the referenced sources. I give consent with lending the thesis.

Prague, Anna Roubíčková

Contents

1 Introduction **6**

2 Basic Concepts **7**
- 2.1 Exposure . . . 7
- 2.2 Colour Spaces . . . 8
- 2.3 Image Blending . . . 10
  - 2.3.1 Manual Method . . . 11
  - 2.3.2 Puzzle Method . . . 11
  - 2.3.3 Pixel Method . . . 11

3 Algorithms Overview **13**
- 3.1 Contrast Blending . . . 13
- 3.2 Radiance Mapping . . . 13
- 3.3 Block Method . . . 14

4 Used Algorithm **15**
- 4.1 Extended Axis of Brightness . . . 15
  - 4.1.1 Placing Images . . . 15
  - 4.1.2 Shifts . . . 16
- 4.2 Weighting Function . . . 16
- 4.3 Average . . . 17
- 4.4 Displaying . . . 18

5 Programmer's Guide **20**
- 5.1 Development Tools . . . 20
- 5.2 Main Decisions . . . 20
- 5.3 Internal Data Structures . . . 21
- 5.4 Implementation . . . 21
  - 5.4.1 Algorithm . . . 21
  - 5.4.2 Fundamentals . . . 23
  - 5.4.3 Command-line Usage . . . 23
  - 5.4.4 Interactive Usage . . . 23

6 User's Guide **25**
- 6.1 System Requirements . . . 25
- 6.2 Instalation . . . 25

6.3 Usage 25
6.3.1 Text Mode 25
6.3.2 Graphic Mode 26

7 Related Work **29**
7.1 Blending in Concurrent Applications 29
7.1.1 Photoshop 29
7.1.2 The Gimp 30
7.2 Evaluation Criteria 31
7.3 Comparison 32

8 Conclusion **34**

Bibliography **36**

Název práce: Combining bracketed pictures
Autor: Anna Roubíčková
Katedra (stav): Katedra softwarového inežnýrství
Vedoucí bakalářské práce: Mgr. Viliam Holub
e-mail vedoucího: holub@nenya.ms.mff.cuni.cz

Abstrakt: Světelný rozsah fotografovaných scén je často větší, než je digitální fotoaparát schopen zachytit. Tento problém bývá řešen několikanásobným vyfotografováním záběru s rozdílnými expozicemi (bracketing). Tato práce představuje program ImageLighter, který automaticky kombinuje sérii fotografií téže scény tak, aby obnovil její původní dynamický rozsah. Při kombinování algoritmus využívá veškerou dostupnou obrazovou informaci. Výsledek by tedy měl zachycovat počáteční rozložení jasu věrněji. Práce popisuje použité algoritmy a datové struktury a poskytuje srovnání s konkurenčními přístupy.

Klíčová slova: dynamický rozsah, bracketing, skládání fotografií

Title: Combining bracketed pictures
Author: Anna Roubíčková
Department: Department of Software Engineering
Supervisor: Mgr. Viliam Holub
Supervisor's e-mail address: holub@nenya.ms.mff.cuni.cz

Abstract: In real-life scenes, the light dynamic range is often larger than a digital camera is able to capture. This problem is addressed by exposure bracketing, i.e. the same scene is captured several times with different exposures. This work presents ImageLighter, a program designed for automatic blending of series of bracketed pictures in order to restore the original dynamic range. The algorithm is designed to use as much visual information as possible while blending. The resulting picture then reflects light levels of the original scene more accurately. The thesis describes algorithms and data structures used. A comparison to other known approaches is presented as well.

Keywords: high dynamic range, bracketing, image blending

Chapter 1

Introduction

Human eye is able to distinguish about 10 million colours. As the common digital cameras work in RGB colour space, which is smaller than human colour space, colours can happen to vary across the scene so much that some areas of the image will appear under- or overexposed no matter what shutter speed or apperture is used. The natural solution is to take several shots with different exposure in order to collect all available visual information and then combine them into one high informative image.

This thesis presents an algorithm for automatic blending of differently exposed pictures of the same static scene. The algorithm was implemented and compared to other applications intended to fulfil the same task.

As the problem originates in insufficient range of digital cameras, the proposed solution is based on reconstructing the range of colours of the original scene. Then, the combination of single shots into one is done by weighted average. This method was chosen because its reducing possible noises or unwanted artefacts. The image with enlarged range of colours is then mapped to standard range to enable its displaying. In contrast to digital chip, the remapping is done in a way minimizing number of saturated and underexposed pixels.

The developed algorithm was implemented in program ImageLighter. The application was designed to be intuitive and fully automatic. Its output is, according to the specification, a high informative image that suffers from under- and overexposure minimally.

ImageLighter is written in C programming language using GTK+ library. The application is designed for platform independent usage—it can be run under Linux and Windows as well as under Mac OS X, if the system has installed GTK+. The architecture of the program allows simple portation of its command-line version to other toolkits than GTK+. On the other hand, porting GUI means to rewrite it fully.

Chapter 2

Basic Concepts

This chapter enlightens basic facts relating to a dynamic range problem. This problem originates in capturing or displaying image on digital device with insufficient number of colours. After defining basic terms (Sect. 2.1, 2.2), fundamental ideas of solution are introduced (Sect. 2.3).

2.1 Exposure

A *Dynamic Range* means the difference between the least and greatest values of an attribute or a variable. In digital imaging, it refers to the span of brightness across a scene. While working with real-life scene, its dynamic range often exceeds the range that device can handle.

The problem is worse when using a digital imaging chip instead of classical film due to their different *Response Function*. It is caused by different sensitivity of digital chip, whose sensitivity deteriorates with decreasing light linearly. Therefore highlights come out better than shadows. On the other hand, using a digital camera simplifies further manipulation with the image.

A *Histogram* is a function of a brightness distribution. The visualisation of a histogram is a column graph where each column represents a value of brightness and its height represents corresponding frequency. An optimal histogram is supposed to be similar to the one shown in Figure 2.1.

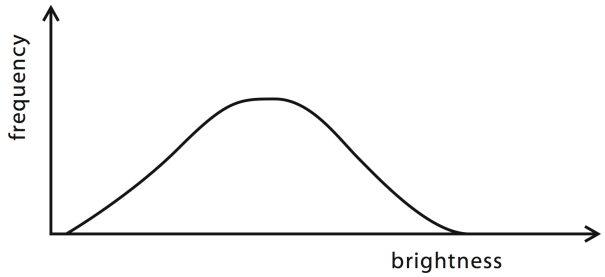


Figure 2.1: Optimal histogram shape

The histogram can be shifted horizontally due to light conditions and camera settings. While the histogram shifts, the range of the device remains the same, which causes deformation of histogram's shape, as it is shown in Figure 2.2. The image is then too dark, with light colours missing; or too light, missing shadows. Thus, a correctly exposed image is an image with a histogram similar to the optimal one.

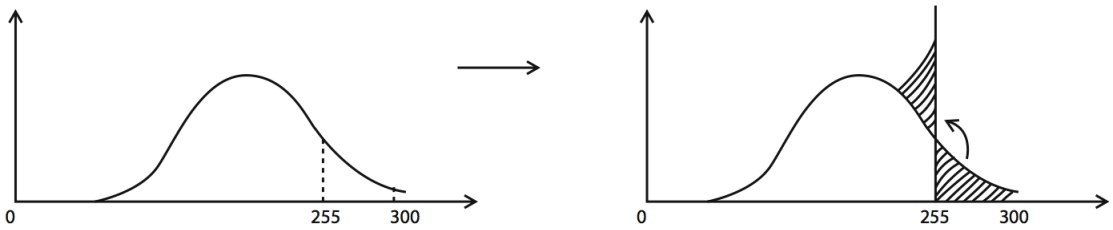


Figure 2.2: Shift and histogram deformation

An *Exposure* is defined as total amount of light that falls on the digital chip (or film) while taking an image. Hence, shift of the histogram on the axis of brightness is closely linked to the exposure.

The exposure can be changed by these three attributes:

- shutter speed, which also influences a motion blur,
- aperture, which also influences the depth of the image,
- and sensitivity of the chip, which influences *graining* of the image

An *Exposure Value (EV)* involves all combinations of attributes mentioned above that result in the same exposure. EV 0 is defined as a value of exposure *acquired* by camera set to exposure time of 1 s and an aperture of f/1.0. The functionality among exposure value (EV), exposure time (t) and aperuture (N) is following:

$$EV = \log_2 \frac{N^2}{t}.$$

Exposure value is a logarithmic function, thus to increase EV by 1 means to double the amount of light falling on the chip. Changing value of EV by 1 is called *Exposure Step*.

2.2 Colour Spaces

A *Brightness* of a pixel is a function of pixel's colour. Therefore, it depends on used colour model because it determines the colour representation. A *Colour Model* is a mathematical abstraction which represents every colour as a serie of numbers. Knowing how to interpret the components, every colour as well as whole colour space related to the model can be evolved.

Human Colour Space

Human colour space is derived from the biological limitations of human eye rather than from mathematics.

Retina in human eye consists of three types of colour receptor cells, each sensitive to light of different wavelengths. As a result, the light is reduced to three colour components. This triplet is called *tristimulus values*.

While it is impossible to stimulate only one cell from the triplet, some tristimulus values are unreachable. The human colour space is defined as a set of all reachable tristimulus values and its size is estimated to be 10 million colours.

RGB Colour Space

The RGB colour model uses additive colour mixing of red, green and blue, which are called *Primary Colours*. Its colour space is represented as a cube with axis belonging to primary colours (Fig. 2.3) and it covers major part of the human colour space.

The absolute brightness counted as an average of R, G and B values does not reflect human's experience of colour brightness. Considering biological characteristics of retina, the brightness computed from RGB values is given by formula

$$Brightness_{RGB} = 0.3R + 0.59G + 0.11B.$$

CMYK Colour Space

The CMYK colour model is similar to RGB model with just a few exceptions. CMYK model uses subtractive mixing of cyan, magenta and yellow colours. To improve production of dark colour tones, the black is added.

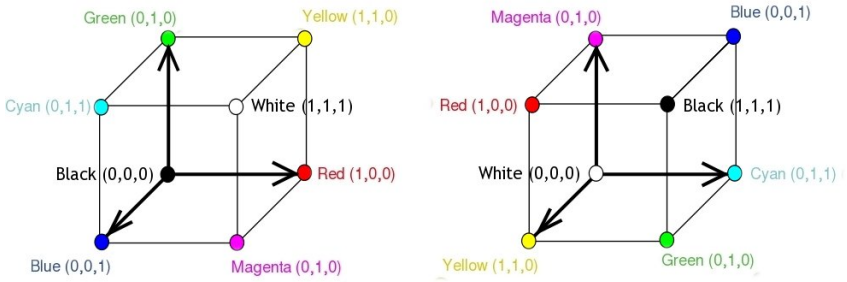


Figure 2.3: RGB colour space and CMYK colour space

HSV Colour Space

The HSV colour space is represented as a cone of colours derived from HSV colour model (Fig. 2.4). The model consists of three components: hue, saturation and value.

Hue refers to the type of colour—whether it is red, blue or yellow. The range of hue is from 0 to 360 and determines angle of rotation on the base of the cone. *Saturation* corresponds to the grayness of the colour. The lower saturation, the more

gray the colour is. Saturation range is from 0% to 100%. *Value* is the brightness of the colour and it is also measured in percents. Given a colour in HSV representation, its brightness is simply

$$Brightness_{HSV} = V.$$

HSL Colour Space

The HSL colour space is double hexcone (Fig. 2.4) and similarly to RGB is not absolute colour space. Therefore colours are not defined exactly, but depends on the shade of primary colours.

The HSL stands for hue, saturation and lightness. The apexes of the double hexcone correspond to black and white. *Hue* is the angular parametr as in HSV model. *Saturation* refers to the distance from the axis and the position on the axis determines the *Lightness*.

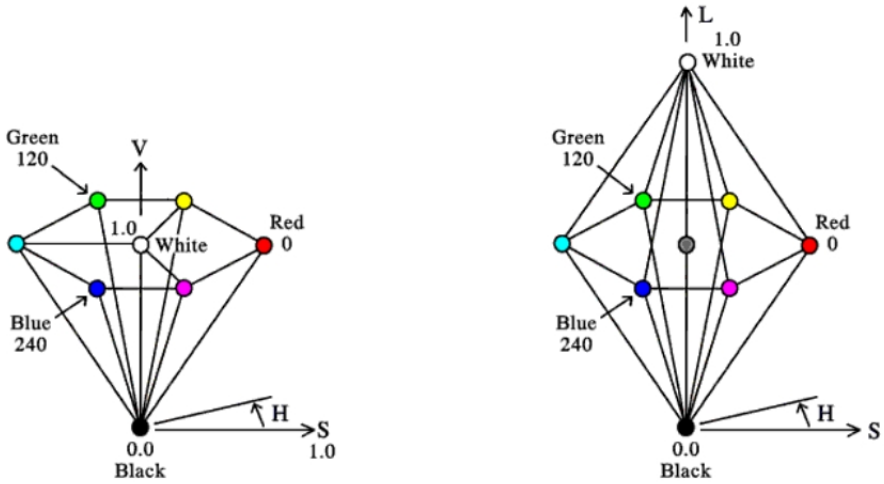


Figure 2.4: HSV colour space and HSL colour space

2.3 Image Blending

Even with the proper combination of the exposure attributes it is sometimes impossible to make a picture of satisfying clearness. The best thing to do then is to take several pictures with different exposures and blend them into one.

Most of today's digital cameras have built-in function called *auto-bracketing*. Photograph can usually choose a number of the exposure steps between folowing pictures and whether three or five pictures will be taken.

A result of the auto-bracketing action is a set of pictures, where every picture differs from the previous one and the following one about constant exposure variance (especially if the camera is placed on tripod and the scene is static). The goal is to retrieve one image where highlights are not burned out and shadows are still clear.

2.3.1 Manual Method

Traditionally, the most reliable and the finest method is to find a human who does the blending manually. Knowing the real-world, human is able to recognize correctly exposed parts of the scene on the source pictures and combine them so that the result looks naturally and clearly. In addition, it is perfectly robust to any shift of pictures. The only problem is that doing so takes one long time, therefore automatic methods are considered.

2.3.2 Puzzle Method

There are correctly exposed areas on every bracketed image. Different images cover different parts of the scene since the exposures varied among single shots. The task is to detect these areas and put them together like a puzzle.

This approach seems to be reasonable: The resulting image will contain only correctly exposed parts, supposing that there were source enough images. Question is how to detect correctly exposed areas and then how to compose them into one.

A pixel is considered to be *correctly exposed pixel* if its brightness lies outside the border values of the dynamic range of the device. Talking about common photography, the closer is the pixel's brightness to 128, the better is the pixel exposed. Likewise, a *correctly exposed area* can be defined as a cluster of pixels from one image whose brightness is close to the center of the range of a chip.

Once the correctly exposed subimages are identified, they can be composed together. However, as the borders of the clusters do not have to fit the borders of real objects, it is complicated to produce naturally looking output.

To avoid forming discontinuities in the image, this method can be transformed to work with all correctly exposed pixels of every image, independently on their relative position. Combining this modified method with an intelligent average reliably removes any extra-edges. However, the result is usually grainy and sometimes even brightness happen to be reversed, as the dark pixels taken from lighter image can be, in the end, lighter than light pixels, that are taken from dark image.

2.3.3 Pixel Method

In order to avoid inconsistencies mentioned above, the problem needs to be reformulated: To each pixel of resulting image should be assigned an appropriate brightness, which approximates its original brightness. That can be achieved by weighting saturated pixels (as well as dark pixels) less than the others. In addition, unnatural artefacts are smoothed owing to the average.

The weakness of methods of this type is sensitivity to any motion or shift among the source images. If the images are not identical, the result will be ghost image. Moreover, if the images on input are unbalanced in brightness (some of them may be extremely light or most of them may be dark), the resulting image will tend to be deviated from optimum.

The average method works as follows:

```
for each position [x,y] of resulting image do
  for i:=1 to number of source images do
    values[i] := colour of pixel [x,y] from i-th image
    weights[i] := weight of pixel [x,y] from i-th image
  enddo
  result [x,y] := weighted_average (values, weights)
enddo
```

Chapter 3

Algorithms Overview

This chapter contains an overview of image blending algorithms. The description presented here serves primarily as an introduction, the strong and weak points of these approaches will be mentioned.

3.1 Contrast Blending

Contrast blending uses mean as blending method and reaches satisfying results [1]. Similarly to weighted average the algorithm works per pixel and in the result all pixels from all exposures are used. The weighting function is more intelligent, though.

Suppose two different exposures of the same scene, long and short, the long one is lighter. The pixels that happen to be burned out in the longer exposure are probably light objects of the real world, so their colour should be taken from the shorter exposure, where they are exposed properly. On the contrary, real dark objects appear too dark in the short exposed image, while they are correctly exposed in the lighter image. In such case, the short exposed pixel should be weighted less than the long exposed one. Hence the lighter the pixel is in the long exposed image, the more weighted it should be in the dark image.

The algorithm uses grayscale of the lighter image as a basis for the weighting function. The grayscale is remapped to interval $\langle 0; 1\rangle$. The weight of a pixel from the darker image is then a number from $\langle 0; 1\rangle$ that corresponds to the brightness of the pixel on the same position in the lighter image.

The algorithm is designed for two differently exposed images, but it can be easily adapted to work with more images. When having more than two images, two darkest are blended. The algorithm then continues with the result and next darkest image, until there are some images left.

3.2 Radiance Mapping

This method consider technical specifications of used camera when trying to reconstruct the original dynamic range of the scene. Having a set of images captured with

different exposures, the chip (or film) response function can be recovered—the response function reflects the relation between amount of light absorbed by the pixel and its colour on the image.

Having enough images and knowing their exposure, the chip response function can be counted [2]. Once the response function is recovered, the radiance map can be constructed—value of every pixel is simply converted (using the response function) to its radiance value.

Doing the conversion of all images and combining products leads to good result. The combination of images is implemented as weighted average, which has two merits: First, pixels with values closer to the middle of the response function can be weighted more than the others, while saturated pixels can be fully ignored. Second, the average helps to reduce noise of the scene. Both these characteristics improve visually the result.

The final radiance map has high dynamic range with pixels' values proportional to their real colours. The image does not suffer from noise, blooming artefacts or saturated pixels while details in the shadows can be still distinguished.

3.3 Block Method

The last mentioned approach [3] is kind of a puzzle method. As the similarly exposed pixels tend to appear in clusters, block method attempts to select the most informative blocks of each image and combine them together to produce highly informative output. The algorithm has two parameters—a block size (d) and a width of the Gaussian blending function (σ)—optimality of their values is critical for the quality of the output.

The first task is to divide inputed images into square blocks of size d. Parameter d is chosen in order to reach the highest possible entropy for every block, as it is tightly linked to the amount of information carried by the subimage—the higher the entropy is, the more information the subimage contains. For every part of the scene, the most informative block is selected among all source images.

The second part of the algorithm is blending selected blocks together. Their simple composing results in image with sharp discontinuities along blocks' borders. Therefore more intelligent blending function has to be used: the colour of a pixel on the resulting image is a product of weighted sum of values of the corresponding pixels from all images. The weighting function is monotonically decreasing and the sum of assigned weights is 1 anywhere in image's domain. Thus, the weighting function is derived from the rational Gaussian function (RaG) centered at the selected block of the image. The RaG is parametrized by σ, the Gaussians width. Estimation of parameters σ and d is based on the gradient-ascent method.

This procedure leads to highy informative images. Unfortunately the result sometimes looks unnaturally. Different choice of parameters influences the output significantly, even the choice done manually can improve the output.

Chapter 4

Used Algorithm

In this chapter, a developed exposure blending algorithm is described. The input of the algorithm is a set of variously exposed shots of the same scene. The task is to create an image exposed correctly in each part of the scene, even if the correct exposures differ.

The proposed solution is kind of a pixel method and the procedure is demonstrated in the Fig. 4.4. First, it attempts to reconstruct the original dynamic range of the scene. Then, every pixel is evaluated by weighting fuction. The evaluation is based on the pixel's quality according to its image and to the image's brightness. The closer to its real value the brightness of the pixel is, the more weighted the pixel will be. The last step is to transform the image for preserving as a low range or for its being displayed on low range devices.

4.1 Extended Axis of Brightness

The problem originates from a disproportion between the range of real-life scenes and the imaging potencial of digital devices. As common image formats use 8-bit colour representation, their range of brightness is $\langle 0; 255 \rangle$. The x-axis of the histogram related to such image is called the *axis of brightness*. Since the dynamic range of real-life scenes is higher than standard for digital images, the x-axis of the real-scene histogram is wider and therefore called the *extended axis of brightness*.

4.1.1 Placing Images

As mentioned above, every inputed image contains correctly exposed areas. These areas represent real-life objects of certain brightness. The real brightness of such objects determines the position of the image on the extended axis of brightness.

If the exposure is higher, the darker real objects are exposed correctly. Due to higher exposure, the image is lighter in average. Consequently, the images with high average brightness should be placed more left on the extended axis of brightness (Fig. 4.1).

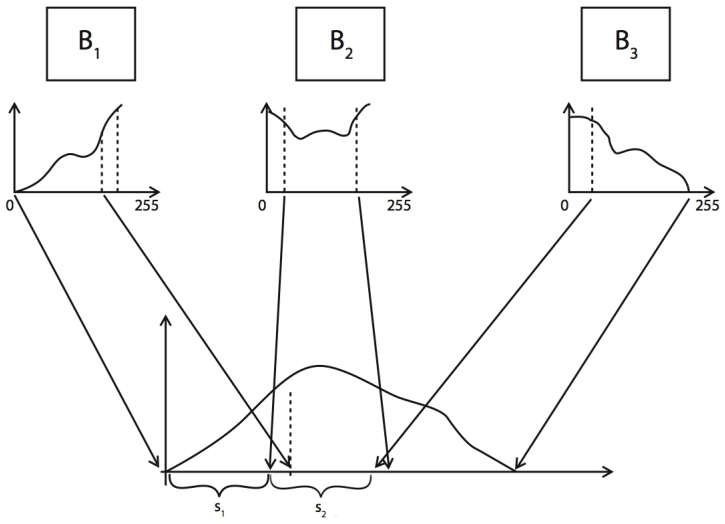


Figure 4.1: Placing images to the extended axis of brightness

4.1.2 Shifts

First, the average brightness of every image is counted as an average of brightness of all pixels in the image. Then the images are sorted in descending order according to the values of their average brightness.

Since the first image represents correctly the darkest objects of the scene, it should cover the lowest part of the extended axis of brightness and it is mapped to $\langle 0; 255 \rangle$, without any shift of its range.

Correctly exposed areas of the following image partly overlaps correctly exposed areas of the first image. An object placed in such area has different colour on each image due to different exposure, even if it is the same object. This difference determines the shift between the pair of images. Denoting $B_{i,i+1}$ the average brightness of correctly exposed part of i-th image that is correctly exposed on (i+1)-st image as well, the shift between them is counted as $\delta_{i,i+1} = B_{i,i+1} - B_{i+1,i}$.

Once the i-th image is shifted, it influences the rest of the images, so they have to be shifted, too. Therefore the final shift (denoted Δ_i) of any image has to reflect previous shifts:

$$\Delta_i = \sum_{j=1}^{i-1} \delta_{j,j+1}.$$

So the range of i-th image is, after its being shifted, $\langle 0 + \Delta_i; 255 + \Delta_i \rangle$.

4.2 Weighting Function

The weighting function is supposed to evaluate every pixel according to the relation between the pixel's value and the real colour the pixel is representing. This relation is determined by pixel's absolute brightness owing to the brightness of the image. The task of the weighting function is to counterbalance the chip response function.

The weighting is performed by function $f(b)$ (Fig. 4.2) defined as follows:

- $w = \frac{1}{2}(\sin(\frac{\pi b}{\alpha} - \frac{\pi}{2}) + 1.01), b \in \langle 0; \alpha \rangle$
- $w = 1, b \in (\alpha; 255 - \alpha)$
- $w = \frac{1}{2}(\sin(\frac{\pi(255-b)}{\alpha} - \frac{\pi}{2}) + 1.01), b \in \langle 255 - \alpha; 255 \rangle$

where w represents the assigned weight, b is the brightness of the pixel and α refers to the critical value that determines whether the pixel is exposed correctly.

The fundamental quality of $f(b)$ is evaluating saturated or underexposed pixels low, but not ignoring them. Therefore the weighting function is placed strictly above x-axis and thus the shift constant is 1.01 instead of 1. The minimal assigned weight is then 0.005. As a result, the average weighted with $f(b)$ reflects more precisely the real colour of the pixel.

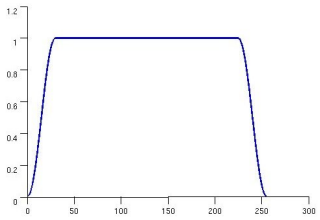


Figure 4.2: Graph of weighting function $f(b)$

4.3 Average

It is now possible to count colour of every pixel on the resulting image. The evaluation is based on colours of the corresponding pixels from the source images. Knowing their weight, a weighted average can be counted.

An *Average*, or a mean, refers to the expected value of a data set. There are several methods how to count the mean and these methods differs from each other as each of them advantages different kind of values. Therefore all of them are implemented and it is up to user to choose the one most fulfilling his expectations.
The notation used below is:

- $\bar{b}$ is the final brightness of the pixel in the output image
- b_i denotes a brightness of the corresponding pixel on the i-th image
- w_i is the weight of the i-th image's pixel
- w is a sum of all weights: $w = \sum_{i=1}^{n} w_i$
- n refers to the number of source images
- $\sum$ abbreviates $\sum_{i=1}^{n}$
- Π stands for $\Pi_{i=1}^{n}$

Overview of means that are implemetned

Arithmetic Mean	$\bar{b} = \frac{\sum(w_i b_i)}{w}$
Geometric Mean	$\bar{b} = \sqrt[w]{\Pi(w_i b_i)}$
Harmonic Mean	$\bar{b} = \frac{w}{\sum \frac{w_i}{b_i}}$
Root Mean	$\bar{b} = \left(\frac{\sum(w_i \sqrt{b_i})}{w}\right)^2$
Qudratic Mean	$\bar{b} = \sqrt{\frac{1}{w} \cdot \sum(w_i b_i^2)}$

4.4 Displaying

The last step to be done is to map the extended dynamic range back to $\langle 0; 255 \rangle$. The extended image is supposed to be exposed correctly, so the shape of its histogram should be proper, only it needs to be compressed.

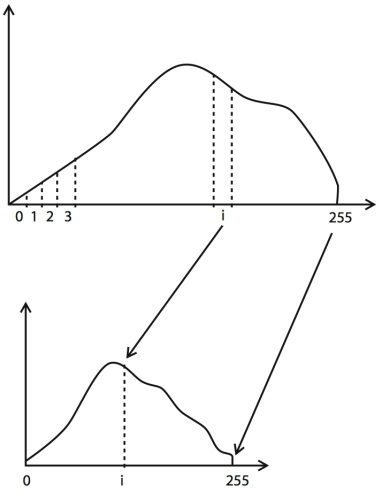


Figure 4.3: Compression of the histogram

The range of the extended histogram is counted as a difference between the darkest and lightest colour used in the scene. In order to lose as little colour tones as possible while compressing, the rare extreme values are not counted in the range.

The range is then divided into 256 intervals of the constant width. Every interval represents an equivalence class and all colour tones from inside are mapped to the same colour in the resulting image.

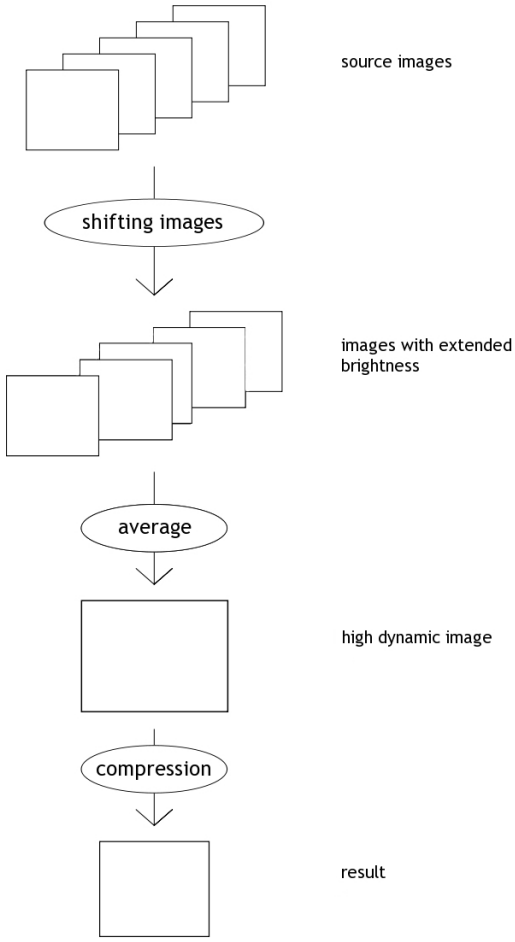


Figure 4.4: Scheme of the algorithm

Chapter 5

Programmer's Guide

This section introduces implementation details of ImageLighter2. Choice of programming language and toolkit are discussed and the description of internal data structures and functions is provided.

5.1 Development Tools

According to the specification, the program is written in C programming language using GTK+ 2.0 library. GTK+ 2.0 is the minimal version ImageLighter2 can be compiled with. Both programming language and library are platform independent and so is the ImageLighter.

5.2 Main Decisions

Colour Model

The average is counted separately for every colour channel in order to detect any possible saturation or underexposure. Therefore the chosen colour model is RGB, even if it seem to be more reasonable to use the HSV or HSI model where brightness is a parameter of colour. However, it would cause complications while detecting saturation of a single colour channel (or two of them).

Data Model

The internal structures as well as whole algorithm are library independent—apart from GUI, the GTK structures are used only for loading images (the first step of the algorithm) and displaying the resulting image (the last step of the algorithm). Therefore the command-line version of ImageLighter is easily portable to other toolkits.

5.3 Internal Data Structures

The data structures and functions for their manipulation are described in a header file `IL2Basics.h`. There is a number of functions providing elementary operations on the image structures mentioned below, such as construction, duplication or conversion between them.

`ImLi2Input`

To preserve images to be blended in unified form regardless of program's mode, the structure `ImLi2Input` is established. It contains an array of pointers to images in GdkPixbuf format, a native GTK+ image format. The number of images is stored inside as well.

`ImLi2Image`

This structure is destined for representing the inputed image and can be constructed from `GdkPixbuf` image representation. It contains image's dimensions and an array of pixels' colours. Every colour is a triplet of values from $\langle 0; 255 \rangle$ representing R, G and B channel. Therefore the type of the array is `guchar*`.

The constructor function responsible for converting image to basic internal representation is `imli2_create_from_pixbuf`, the conversion back to `GtkImage` is done by `imli2_gtk_show` function.

`ImLi2LongImage`

This structure represents an image on the extended axis of brightness. While the extension of the axis of brightness is caused by increase of the range of brightness, the structure contains an array of `gint*` type representing colours, then the image's dimensions and its shift on the extended axis of brightness.

The constructor function is `imli2_long_shift`. The structure can be converted back to basic representation by function `imli2_convert_long_to_short`.

5.4 Implementation

The program can be launched in graphic or text mode. The modes differs only before starting the blending algorithm, as it is described in the User's Guide. Both modes create indentical structure of inputed images (`ImLi2Input`) and pass it to the `imli2_process` function together with the average function to be used.

5.4.1 Algorithm

The function `imli2_process` controls the whole blending algorithm. After transforming the inputed images to the basic internal representation, the images are sorted by `qsort` according to their average brightness.

Their placing onto the extended axis of brightness is implemented as a conversion from `ImLi2Image` to `ImLi2LongImage`. To do the transformation of image, its shift needs to be known. The shifts are counted by `imli2_count_differences` as it is described in section 4.1.2.

Weighting the pixels is done by `_weight_func_sin`, which is described in section 4.2 in detail. There are several functions counting different kinds of average and each of them refers to a method mentioned in section 4.3. Namely the functions are:

- `imli2_arithmetic`
- `imli2_geometric`
- `imli2_harmonic`
- `imli2_root`
- `imli2_quadratic`

The function `imli2_convert_long_to_short` performs the final conversion back to standard byte representation.

In order to reach better results, two more functions are run during blending. Just after the first transformation (to the `ImLi2Image` representation) every image is passed to `imli2_reduce_noise` which purifies it from Gaussian noises. Then, the edges are detected by `imli2_detect_edges` to avoid forming of abnormalities along them.

Functions related to the algorithm are gathered in header file `IL2Algorithm.h`.

Noise Reduction

A *Noise* is an unwanted component of an image which restrain a quality of details on the image. Noise reduction is the process of removing the noise from the image. As there are different kinds of noise [4], different filters can be used.

The algorithm uses Gaussian Filter, which is based on matrix convolution. The image, as a matrix of pixels' brightness, is convolved with the Gaussian mask (Fig. 5.1). That transforms every pixel's value closer to the value of its neighbours.

$$\begin{pmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{pmatrix}$$

Figure 5.1: Gaussian mask

The disadvantage of this filter is slight blurring of the edges in the scene, but that doesn't cause any significant problems.

Edge Detection

The *Edge* represents a border between objects of the scene and appears as a sudden change of the colour. The more sudden the change is, the more likely it represents an edge. The detection of rate and direction of colour change (gradient) in each pixel can help identify edges.

The Sobel operator[5] consists of two matrices (Fig. 5.2) and calculates the gradient vector: the first matrix detects horizontal changes and the second one vertical changes. Other directions are derived from the join of the operator matrices.

$$\begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \quad \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}$$

Figure 5.2: Two matrices of Sobel operator

In each pixel, the gradient vector points in the direction of the most significant colour change. Convolution of Sobel operator and the image results in a zero vector in the areas of low colour changes or in a vector pointing across the edge.

5.4.2 Fundamentals

A set of functions used in program as supplemental is placed in `IL2Fundamental.h`. Mathematical and other constants as well as global declaration of variables are presented in `IL2Constants.h`

The initialization of GTK+ library together with basic functions important for working with widgets are declared in `IL2Gtk.h` file. These functions are used by graphic mode and text mode, too, as the text mode displays the blended image as well.

5.4.3 Command-line Usage

The functions related to processing the textual input and filling in the `ImLi2Input` structure can be found in file `IL2Cmdl.h`.

The essential function of this part of the program is `imli2_read_input`. Its task is to derive from the input the list of images to be blend and to identify options. The options can influence run of the blending algorithm, as it is described in User's Guide.

5.4.4 Interactive Usage

The program is operated interactively in the graphical mode. Thus the program is event driven, which is realized through method of signals and their handling. The functions responsible for handling singnals are declared in `IL2Gui.h`. `IL2Tree.h` contains supportive functions for displaying the navigation tree in top left corner of application window.

The signals used for controlling the graphic interface are mostly oriented to handle clicking buttons. However, the important group is formed by signals related to Drag-and-Drop actions between `scene_viewport` and `combine_viewport` widgets, which is essential for selection of images for blending. These functions are declared in `IL2Signals.h`.

Chapter 6

User's Guide

6.1 System Requirements

The program needs GTK+ 2.0 library (or higher) and standard C libraries to run. In Mac OS X, it is necessary to have installed the X Window System, too.

6.2 Instalation

The program is distributed as a source file package. In order to compile the application, the GTK library properly installed on the system is required. The compilation is done by running `make` in the folder `ImageLighter/install`. The program is installed in the current user's home directory (`/imagelighter2/ImageLighter2.out`).

6.3 Usage

The application can be launched from command-line as well as via icon. The program will start in interactive graphic mode when launched without arguments.

6.3.1 Text Mode

ImageLighter runs in text mode when launched from command line with arguments. It is necessary to specify images that should be blended. Optionally, dimensions of the resulting image can be specified and if the output should be saved, it is necessary to insert desired name.

Launching

The application is launched from its folder using command:

```
./ImageLighter2.out [-s name.suf -d width height] images_to_be_blended
```

Options

`-s name.suf`
save the result as `name.suf`, where `suf` determines the image format of the result

`-d width height`
determines dimensions of the result

6.3.2 Graphic Mode

Launching

ImageLighter can be launched in graphic mode double-clicking its icon or using command:

`~/imagelighter2/ImageLighter2.out`

Application Window

The application window consists of controlling and displaying part. The controlling part is divided into two: navigation section and section for controlling options and actions related to blending process. The displaying part has two sections as well - the upper section displays previews of images present in a chosen folder, and the lower sections contains images selected for blending.

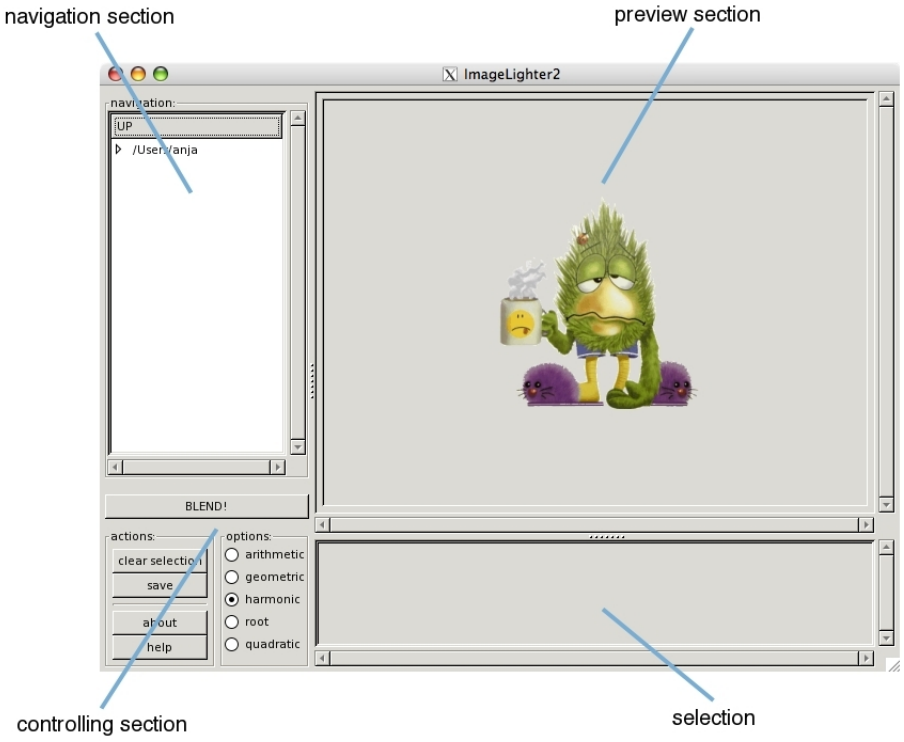


Figure 6.1: Parts of application window

Options

The only option to be set is kind of mean to be used. A mean needs to be set before starting the blending process.

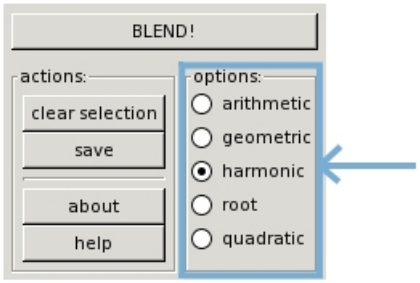


Figure 6.2: Setting mean

Choice of Images

First, the appropriate folder has to be chosen in the navigation tree. At the beginning, the root of the navigation tree is user's home directory. Single-click on the triangle before directory's name expands corresponding directory, while single-click directory's name chooses it and previews of contained images are shown. Double-click makes clicked directory the new root of the navigation tree. Clicking the button "UP" moves the navigation tree root to its parent dictionary.

Once the directory is chosen, the images contained in it are shown in the upper section of displaying part of the window. The images chosen for blending are specified by placing them into the lower section. This can be done by dragging them from the upper part and dropping in the lower section.

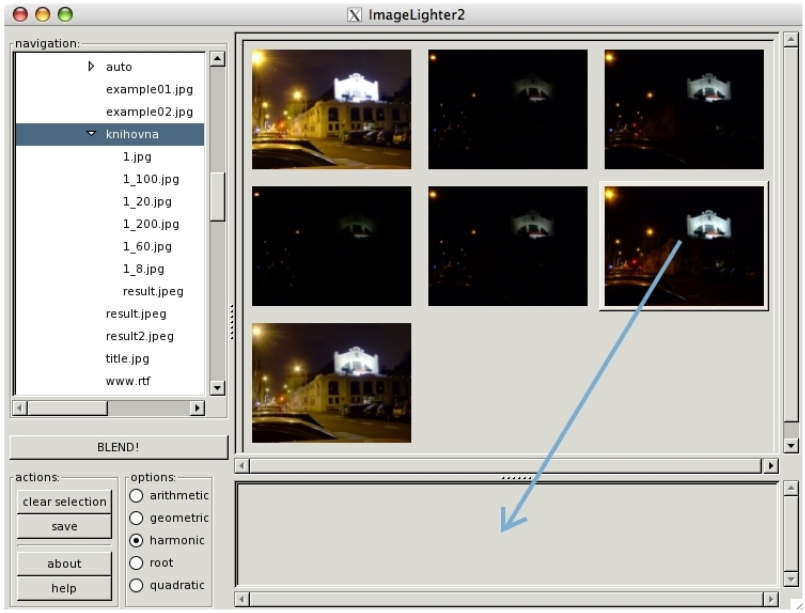


Figure 6.3: Adding images to the selection

To view the image in detail, clicking the image's preview enlarges it. To see the preview of images in the folder again, it is necessary to choose the folder in the navigation tree again.

Single pictures can be removed from selection by drag-and-drop in opposite direction. Removing whole selection is done by clicking the button "clear selection".

Figure 6.4: Removing images from the selection

Actions

After choosing images, the blending action is launched clicking the button "BLEND!". The resulting image shows in the displaying part of the application window. Then it can be saved.

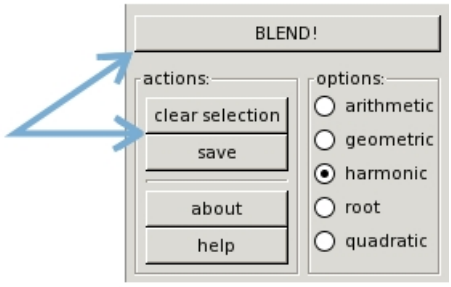


Figure 6.5: Action buttons

Chapter 7

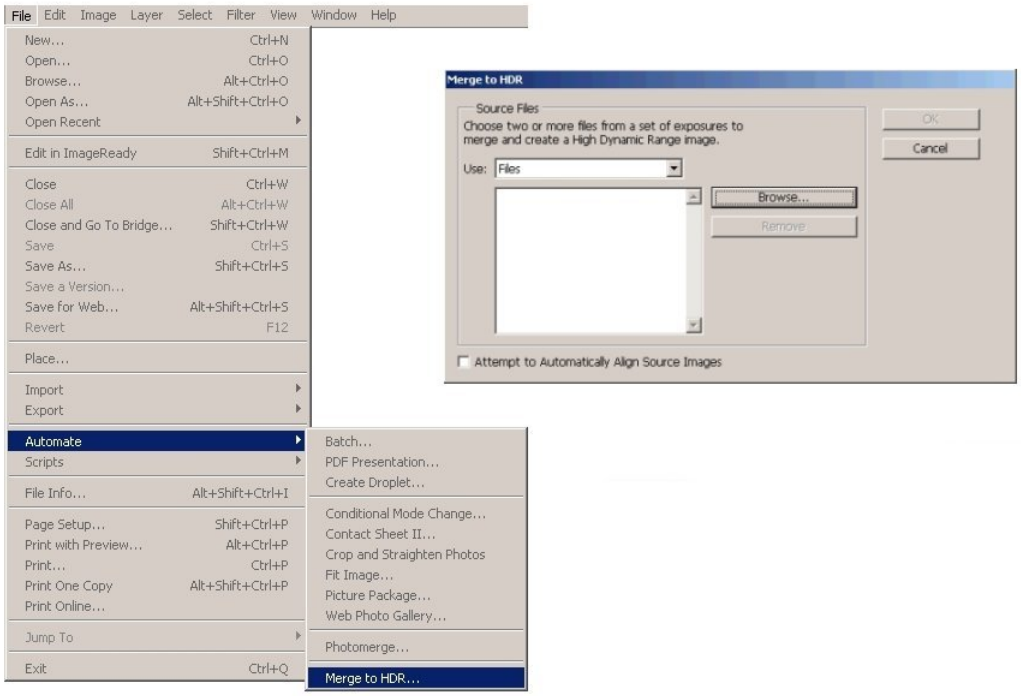
Related Work

As the problem of insufficient range of digital cameras rises for their increasing usage, a few softwares were developed to help photographers counterbalance it. The most common ones—commercial as well as open-source—are introduced in this chapter and confronted to ImageLighter.

7.1 Blending in Concurrent Applications

7.1.1 Photoshop

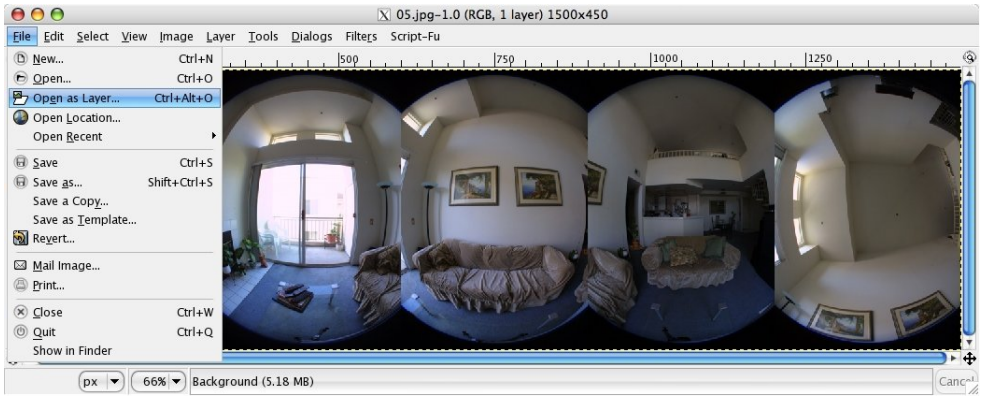
Photoshop, since Creative Suite 2 pack, contains function "Merge to HDR" that automatically does the blending of given images. It can be accessed either from menu (*File* → *Automate*) or from Bridge (*Tools* → *Photoshop* → *Merge To HDR*):



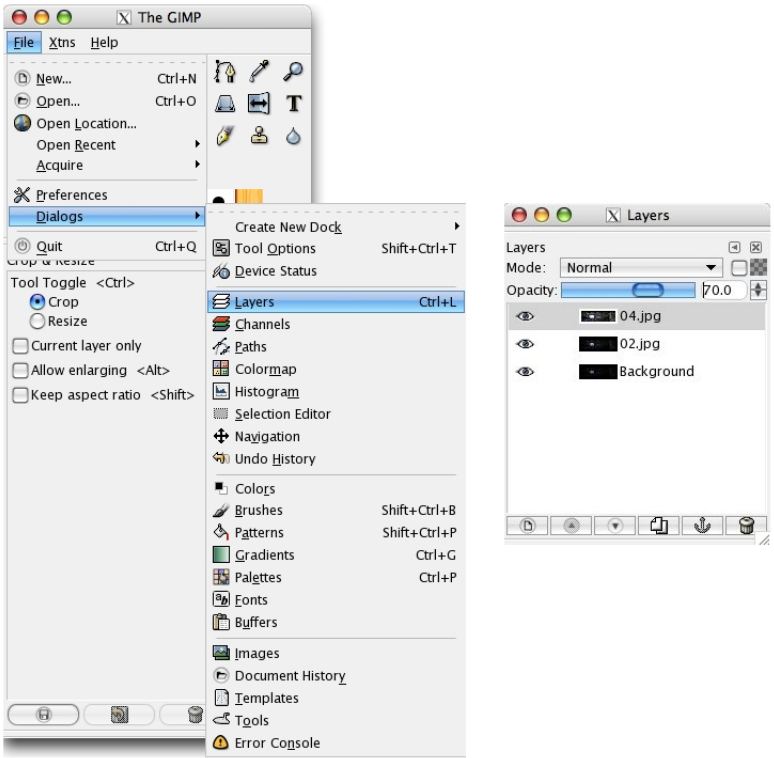
7.1.2 The Gimp

The Gimp does not yet involve such automatic function as Photoshop. So the only way to blend images with Gimp is manually:

The darkest image is opened as usual and every following image is opened as a layer of the first one (*File* → *Open as Layer*).



Each layer's opacity has to be set then. The proper choice of opacity is the essential step of manual blending process. The opacity can be set in the layers dialog (*File* → *Dialogs* → *Layers*).



7.2 Evaluation Criteria

In order to evaluate various software they were run with the set of images to confront qualities of their outputs. Their price, time consumption and work difficulty is considered as well.

Input

The photographies used as the test serie (Fig. 7.1) were taken with following camera settings (top to bottom, left to right): $\frac{1}{6}$s, $\frac{1}{3}$s, $\frac{7}{10}$s, $\frac{3}{2}$s, 3s, 6s, 10s, 20s.

Figure 7.1: Source images[6]

Quality of the Output

There are at least two critical areas on the test scene (Fig. 7.2): the television in the left part and the house outside the glass door in the second part of the scene. The screen of the television remains too dark until the rest of the scene is overexposed, whereas the house is hardly visible until the rest of the scene becomes underexposed. Examining these two areas together with the overall lightness of the scene, the quality of the output can be discussed.

Figure 7.2: Critical areas of the scene

7.3 Comparison

The full version of Adobe Photoshop CS3 costs approximatelly 830 EUR ($\sim$ 24.000 CZK). By contrast to Photoshop, Gimp is free software distributed under GNU LGPL licence. The ImageLighter is free as well.

Knowing location of the "Merge To HDR" function in Photoshop, the blending can be done quickly. During the blending process several settings can be changed in order to improve the result which, for ordinary user, can be a bit confusing. Fortunately, the default settings work quite good.

The usage of Gimp is quite intuitive, but the manual blending is time-consuming. In addition, the acceptable result is extremely difficult to be reached without previous experience in blending.

ImageLighter is designed wholly for blending images. The blending function is accessible directly in the main application window, so the usage is simple, fast and fully automatic, requiring no extra skills.

Probably the most important part of the comparison is the resulting image. All presented images (Fig. 7.3) are outputs of the applications with default settings if there was a choice. These images could be probably improved by slight colour correction, but its quality would depend on user's skills more than on the quality of the software.

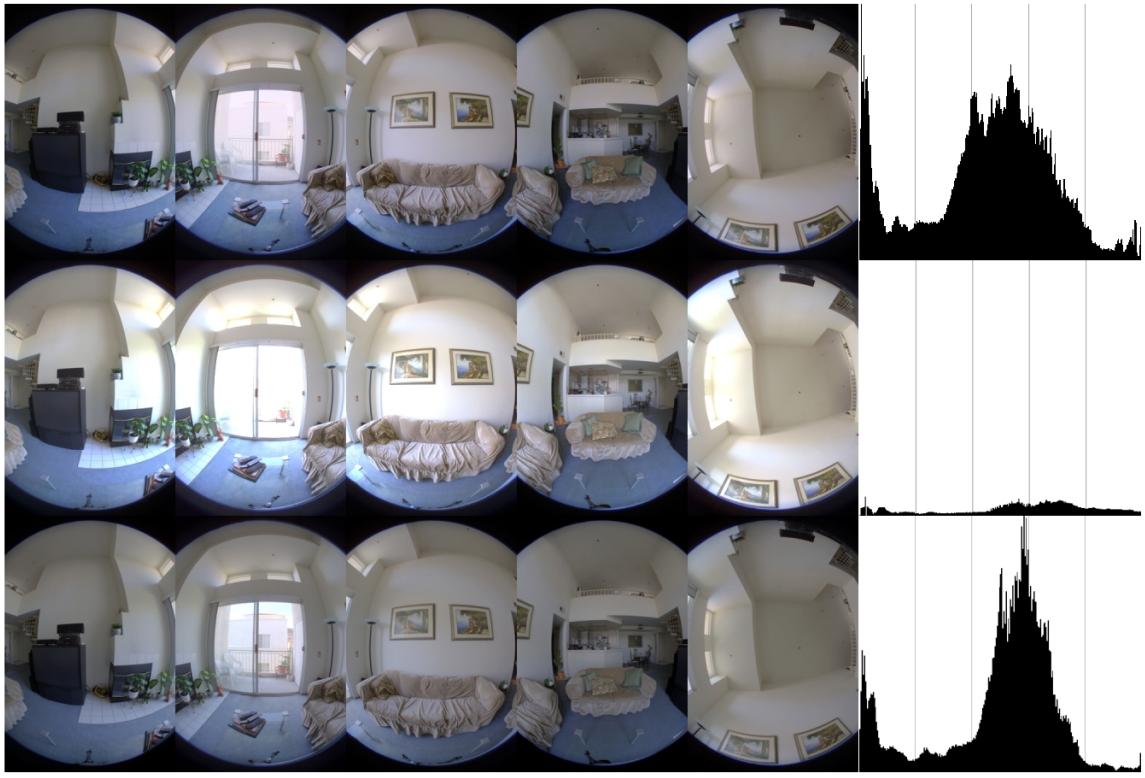


Figure 7.3: The results (top to bottom) from Gimp, Photoshop and ImageLighter

The output of the Gimp satisfies requirements—the colours across the scene are natural and the image is more informative than any of the inputed images. Every part of the scene is clear, especially there is recognizable screen of the television and the house outside is visible, too. The shape of the scene's histogram approximates the optimal shape.

Figure 7.4: Product of The Gimp

Image developed by Photoshop is not so clear, though. Using the default settings is simple, but the result seems to be localy too light. Even if colours and whole image look natural, the image misses available visual information, as the house outside is nearly invisible. Setting some special options during the blending would provide better result at the expense of spent time and user's comfort.

Figure 7.5: Product of Photoshop

The image blended by ImageLighter2 is fully informative—both the television screen and the house outside are perfectly visible. The house is even more clear than the one on the Gimp image. On the other hand the colours of the scene tend to be a little more grayish. General feeling of the image is therefore pale, but accurate.

Figure 7.6: Product of ImageLighter2

Chapter 8

Conclusion

The aim of this work was to develop and implement an image blending algorithm that would produce more quality images than the source ones. The quality of image was understood as the measure of information contained by the image, which links closely to the optimality of its histogram.

The idea of proposed solution is reconstruction of the original high dynamic range. The weighted average is used for combining the images in order to smooth any possible noises and artefacts. In order to compare designed algorithm to other possible approaches, the application ImageLighter was developed. It is fully automatic and multiplatform solution for image blending.

The algorithm proved itself to be competitive compared to other graphic software: Even though manual output of Gimp is approximately same quality, Gimp provides more graphic tools to correct colours of the image. On the other hand ImageLighter is as automatic as possible—user only specifies images to be blended. The result may be improved with colour levels correction, but usually it isn't necessary.

Compared to Photoshop, ImageLighter's big advantage is price. Moreover, "Merge to HDR" function run with default settings result in naturally looking images, that aren't as informative as possible. Helping Photoshop a little while blending, the result can be nicer than the one from ImageLighter. Similarly to Gimp, Photoshop provides more tools for image adjustment than just their blending, so the images can be easily manipulated and improved in comparison with ImageLighter.

There are few possible improvements to be done in future. First, and probably the most useful one, is the correction of movement among the single shots. Currently, ImageLighter requires images that are identical except the exposure. Any shift or movement on the scene results in a ghost image.

This is an important feature, as sometimes even shots captured with camera placed on tripod differs from each other. The problem is that (besides linear shifts or rotation) the camera can happen to move so that the angles and distances deform on the image. Those movements are not only difficult to be detected but hard to correct as well.

If the function doing the correction of dislocations was available, it could be easily included into the algorithm. This function would be simply run during the algorithm any time before counting final average.

As ImageLighter is a single-thread application, the graphic interface is not refreshed during the process of blending. In future, special thread for the calculation itself should be added.

Among further improvements to be done belongs implementation of tools allowing some basic colour transformations, for instance modification of gamma curve or adjustment of colour levels.

Bibliography

[1] Tina Dong, Sufeng Li, Michael Lin: *High Dynamic Range Imaging for Display on Low Dynamic Range Devices*, Stanford School of Engineering, 2006.

[2] Paul E. Debevec, Jitendra Malik: *Recovering High Dynamic Range Radiance Maps from Photographs*, University of California at Berkeley, 1997.

[3] Arthur A. Goshtasby: *High Dynamic Range Reduction Via Maximizaton of Image Information*, Wright State University, 1985.

[4] Al Bovik: *Handbook of image and video processing*, Academic Press (2000), 325—335.

[5] Bill Green: *Edge Detection Tutorial*, `http://www.pages.drexel.edu/~weg22/edge.html`

[6] Erik Krause: *Contrast Blending*, `http://www.erik-krause.de/blending`